Craig Meinschein

Brianna Buckner

CS 350 Project Proposal

A Comparison of Sorting Algorithms

By Craig Meinschein and Brianna Buckner

Project Description

For our project, we plan to implement and evaluate the performance of three different sorting algorithms which are considered to be among the most efficient for sorting large lists of elements. The three algorithms that we are interested in are Heapsort, Mergesort, and Quicksort. We chose sorting algorithms as our topic because we found it the most interesting out of the offered suggestions. They are also extremely useful in everyday coding; there are a lot of circumstances where you need to sort a collection of data, and there are rarely standard libraries available for various sorting algorithms.

Our plan for collaborating as a group on this project is to create a shared Git repository for holding and managing our code. We will divide the sorting algorithms between us and implement them independently as much as possible, in order to parallelize our development. We will review each other’s code in our Git repository as it is being developed, and after each algorithm’s implementation is finished. We will try to standardize our implementation so that the same tests can be run on all three algorithms, to ensure that the results are unbiased by special implementation outside of the algorithm itself.

Implementation Details

In order to thoroughly test the time-efficiency of these algorithms, we will prepare several small, large, and very large lists of integers. These lists will have a variety of characteristics which may affect the performance of sorting, such as being partially-sorted, reverse-sorted, completely random, and having many (or few) duplicate elements. We will then use each algorithm to sort these lists while measuring the amount of time it takes for each one to execute. We will then perform an analysis of the data we gather and compare raw execution times among the algorithms, and proportional time with respect to the number of elements, to see if the numbers approximate the expected efficiency.

Our implementation language will be Python. We chose Python because it is easier to write code in, which means we can focus on ensuring that our implementation is correct and on writing a wide variety of tests.

Report Features

The features that we plan to include as part of our report are as follows:

- Raw aggregated and averaged time data on execution times for each sorting algorithm.
- Execution time with respect to the number ‘n’ elements to be sorted for each sorting algorithm.

- Comparison of best case, worst case, and average case execution times for each sorting algorithm individually, as well as against the other algorithms.
- Comparison of best case, worst case, and average case execution times for each sorting algorithm against expected values given by their big-O complexity.
- Graphs of time data for the algorithms based on the size of the input and characteristics of the input lists.

Timeline

Our planned timeline is as follows:

Week of 23rd May

- Implement Mergesort and verify correctness.
- Assemble/generate test data to be sorting.
- Implement code to collect metrics and data about execution time in Mergesort.

Week of 30th May

- Implement Quicksort and verify correctness.
- Implement Heapsort and verify correctness.
- Implement code to collect metrics and data about execution time in Heapsort and Quicksort.

Week of 6th June

- Final code review and optimization.
- Run each sorting algorithm and gather data.
- Analyze data, draw conclusions, and write final report.